**什么是函数指针？**

指针前面已经写过好多，自以为认识的差不多了，但是今天突然看到一个问题，写一个函数指针，函数指针就是一个指向函数首地

址的指针，接下来写几个指针：

写一个函数指针

写一个函数指针数组

写一个指向函数指针数组的指针

看到这三个是不是觉得挺绕~

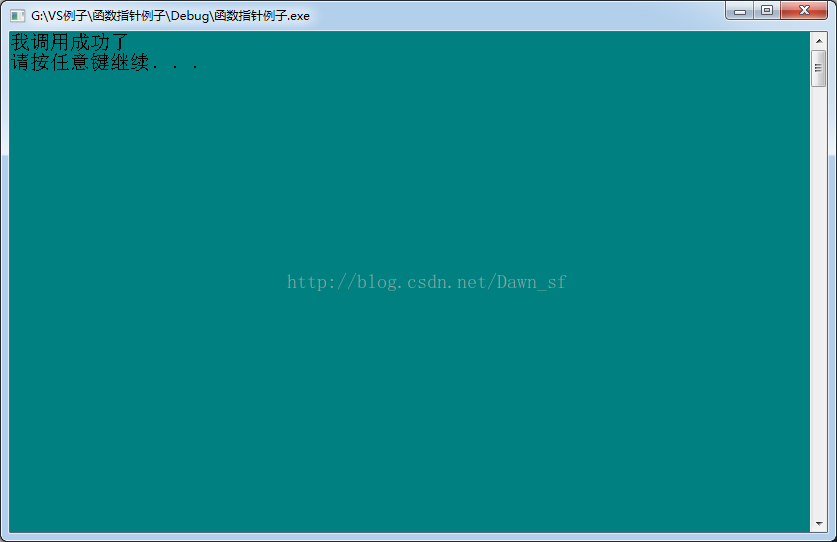
**==》定义函数指针**

举个例子看以下代码:

**[cpp]** [view plain](https://blog.csdn.net/dawn_sf/article/details/55049524) [copy](https://blog.csdn.net/dawn_sf/article/details/55049524)

1. #include<stdio.h>
2. #include<Windows.h>
3. **void** fun()
4. {
5. printf("我调用成功了\n");
6. }
7. **int** main()
8. {
9. **void**(\*pfun)() = &fun;
10. (\*pfun)();
11. system("pause");
12. **return** 0;
13. }

运行结果是什么呢？



这个就是一个函数指针的声明。

**==》如何定义一个函数指针数组？**

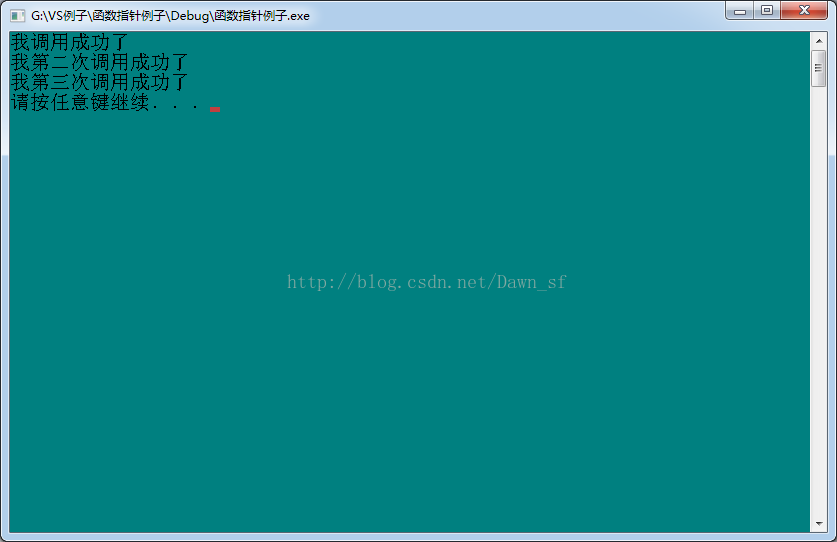
以前我写过一个博客介绍数组指针和指针数组的区别，我们应该知道函数指针数组，是每一个元素为函数指组的数组，对它是数组。

**现在来看看他的用法：**

**[cpp]** [view plain](https://blog.csdn.net/dawn_sf/article/details/55049524) [copy](https://blog.csdn.net/dawn_sf/article/details/55049524)

1. #include<stdio.h>
2. #include<Windows.h>
3. **void** fun1()
4. {
5. printf("我调用成功了\n");
6. }
7. **void** fun2()
8. {
9. printf("我第二次调用成功了\n");
10. }
11. **void** fun3()
12. {
13. printf("我第三次调用成功了\n");
14. }
15. **int** main()
16. {
18. **void**(\*pfun)() = &fun1;
19. **void**(\*pfun2)() = &fun2;
20. **void**(\*pfun3)() = &fun3;
21. //接下来就是定义一个数组把他们三个装进去。
22. **void**(\*pfunarr[3])();
23. //在这里呢看着个指针数组，他就是一个具有3个以函数指针为内容的元素的函数指针数组。
24. pfunarr[0] = &pfun<span style="font-family:Arial, Helvetica, sans-serif;">;</span>
25. pfunarr[1] = &pfun2;
26. pfunarr[2] = &pfun3;
28. pfunarr[0]();
29. pfunarr[1]();
30. pfunarr[2]();
31. system("pause");
32. **return** 0;
33. }

那我们一起来看看运行结果



很多人疑惑那个声明是怎么写出来的，如果你定义一个整形数组呢？  int arr[3]。那么我们刚刚知道函数指针类型 怎么写？

void (\*pfun)() 这个函数指针类型想把它变成数组就去找那个名字往它后面加一个[3]  就ok了~ 也就是   void(\*pfun[3])(); 

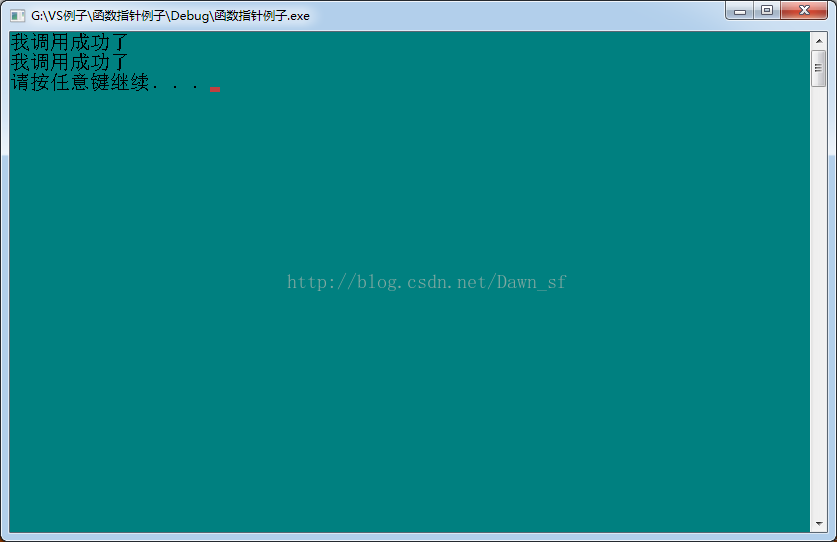
**==》定义一个指向 指针数组的指针**

上面我们已经发现规律，那我再看如下代码~

**[cpp]** [view plain](https://blog.csdn.net/dawn_sf/article/details/55049524) [copy](https://blog.csdn.net/dawn_sf/article/details/55049524)

1. #include<stdio.h>
2. #include<Windows.h>
3. **void** fun1()
4. {
5. printf("我调用成功了\n");
6. }
7. **void** fun2()
8. {
9. printf("我第二次调用成功了\n");
10. }
11. **void** fun3()
12. {
13. printf("我第三次调用成功了\n");
14. }
15. **int** main()
16. {
18. //void(\*pfun)() = &fun1;
19. //void(\*pfun2)() = &fun2;
20. //void(\*pfun3)() = &fun3;
21. //接下来就是定义一个数组把他们三个装进去。
22. **void**(\*pfunarr[3])();
23. **void**(\*(\*pfunarr2)[3])();
24. //在这里呢看着个指针数组，他就是一个具有3个以函数指针为内容的元素的函数指针数组。
25. pfunarr[0] = &fun1;
26. pfunarr[1] = &fun2;
27. pfunarr[2] = &fun3;
29. pfunarr2 = &pfunarr;
30. (\*pfunarr2)[0]();
31. pfunarr[0]();
32. system("pause");
33. **return** 0;
34. }

再看看结果~



**我们把那个声明拿下来 分析一下**

**void (\*(\*pfunarr2)[3])();**

**首先呢，把  （\*pfunarr2）[3]  这一段代码先去掉。**

**（\*pfunarr2）[3] 这段代码很明显就是一个指针数组的原型。**

**那么它剩下来的部分是 void (\*)() 这个是一个指向函数的指针原型。**所以他两个拼在一起 

**void (\*(\*pfunarr2)[3])();**

**就叫做指向函数指针数组的指针。**

**这里这些听起来特别绕，但是呢你自己理解了，它们拿出来其实就都挺简单的喽。**

<https://blog.csdn.net/dawn_sf/article/details/55049524>

使用函数指针的好处在于，可以将实现同一功能的多个模块统一起来标识，这样一来更  
容易后期的维护，系统结构更加清晰。或者归纳为：便于分层设计、利于系统抽象、降低耦合度以及使接口与实现分开.

上面的话是有道理的。只是，要真的体会这些名词，需要你亲历躬行，一个“[百度知道](https://www.baidu.com/s?wd=%E7%99%BE%E5%BA%A6%E7%9F%A5%E9%81%93&from=1012015a&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YLPvcYnywWPjIhP1mdnWfz0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPjcznHTvrjD3)”是很难让你真正体会并理解的。国人一些教科书也不负责任，人云亦云的说这些话，但很少能透彻讲解。我小试一下。  
  
1. 便于分层设计：[函数指针](https://www.baidu.com/s?wd=%E5%87%BD%E6%95%B0%E6%8C%87%E9%92%88&from=1012015a&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YLPvcYnywWPjIhP1mdnWfz0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPjcznHTvrjD3)是引用，是间接层，或曰[隔离层](https://www.baidu.com/s?wd=%E9%9A%94%E7%A6%BB%E5%B1%82&from=1012015a&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YLPvcYnywWPjIhP1mdnWfz0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPjcznHTvrjD3)。它输出到上层，给上层用户用。函数实体是实现，在下层，给开发者用，实现者（软件工程师）关注。这就是简单的分层的概念了。上层用户想让一个函数所做的东西会变化时，我们只需要改变底层实现，并用函数指针指向新的实现就行了。  
再精炼一下分层：分层的核心是对接口进行设计和实现。函数指针的作用就是提供不同实现的统一接口。  
2. 利于系统抽象：只有存在多个类似的实体需要模拟、操作或控制时（这种情况很多）才需要抽象。多个类似的实体就是对象，抽象的结果就是类。在C里边，可以用函数指针数组完成这种抽象。如， fopen 就是一个例子。他可以打开文件。C里面将磁盘文件、[串口](https://www.baidu.com/s?wd=%E4%B8%B2%E5%8F%A3&from=1012015a&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YLPvcYnywWPjIhP1mdnWfz0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPjcznHTvrjD3)、[USB](https://www.baidu.com/s?wd=USB&from=1012015a&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YLPvcYnywWPjIhP1mdnWfz0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPjcznHTvrjD3)等诸多设备抽象为文件。  
3. 降低[耦合度](https://www.baidu.com/s?wd=%E8%80%A6%E5%90%88%E5%BA%A6&from=1012015a&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YLPvcYnywWPjIhP1mdnWfz0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPjcznHTvrjD3)以及使接口与实现分开：第1条中的解释已经说明了这一点。  
再具体一下：  
我曾搭建过一个[嵌入式](https://www.baidu.com/s?wd=%E5%B5%8C%E5%85%A5%E5%BC%8F&from=1012015a&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YLPvcYnywWPjIhP1mdnWfz0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPjcznHTvrjD3)平台。其中的设备操作（硬件驱动）采用了多组函数指针数据，并进行了简单的[封装](https://www.baidu.com/s?wd=%E5%B0%81%E8%A3%85&from=1012015a&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YLPvcYnywWPjIhP1mdnWfz0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPjcznHTvrjD3)，其结果是：  
业务软件使用函数指针数组的[封装](https://www.baidu.com/s?wd=%E5%B0%81%E8%A3%85&from=1012015a&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YLPvcYnywWPjIhP1mdnWfz0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPjcznHTvrjD3)函数访问设备。这个[封装](https://www.baidu.com/s?wd=%E5%B0%81%E8%A3%85&from=1012015a&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YLPvcYnywWPjIhP1mdnWfz0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPjcznHTvrjD3)可展示为：  
int DevOpen（char \*strDevName）;  
int Write(int DevID, char\* DataFrom, int StartDevAddr, int DataLong);  
int Read(int DevID, char\* DataTo, int StartDevAddr, int DataLong);  
int DevClose（int DevID）;  
  
这组抽象出来的函数，是一组语义清晰且稳定的上层接口，为上层的业务开发团队使用。底层的驱动层，各自实现自己的设备打开、读、写及关闭代码，然后，注册自己的设备到系统列表里。上层业务接可以使用了。期间的耦合在这里：  
int Write(int DevID, char\* DataFrom, int StartDevAddr, int DataLong)  
{  
/\* Error detecting. \*/  
return DevWrite[ DevID ]( char\* DataFrom, int StartDevAddr, int DataLong );  
}  
  
其中，DevWrite[ DevID ]就是一个函数指针数组：  
int （\*DevWrite）[ MAX\_DEV\_NUM ]( char\*,int,int ) = {0};  
里面放的就是多个设备的写的操作。其中包括：[UART](https://www.baidu.com/s?wd=UART&from=1012015a&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YLPvcYnywWPjIhP1mdnWfz0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPjcznHTvrjD3)，[RTC](https://www.baidu.com/s?wd=RTC&from=1012015a&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YLPvcYnywWPjIhP1mdnWfz0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPjcznHTvrjD3" \t "_blank)，WatchDog。EEPROM，DigitalInput，DigitalOutput，Key,LCD，LED。  
好了。说的太多了。如需要学习，[可邮件livefuture@163.com](mailto:可邮件livefuture@163.com)。

https://zhidao.baidu.com/question/422176818.html